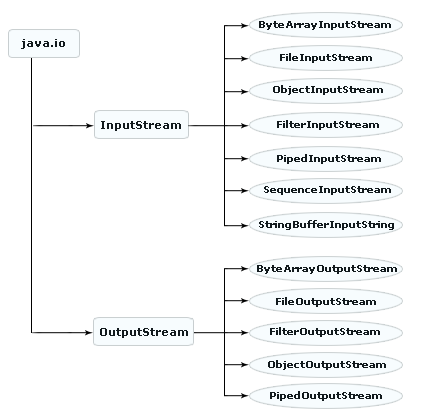
**Java I/O and Streams**

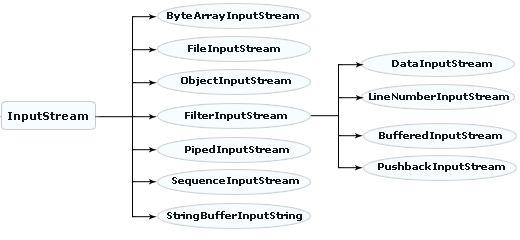
**Introduction:**

* The Java Input/output (I/O) is a part of **java.io** package. The **java.io** package contains a relatively large number of classes that support input and output operations.
* The **InputStream** and **OutputStream** are central classes in the package which are used for reading from and writing to byte streams, respectively.
* The java.io package can be categories along with its stream classes in a hierarchy structure shown below:



**InputStream:**

* The **InputStream** class is used for reading the data such as a byte and array of bytes from an input source.
* An input source can be a **file**, a **string**, or **memory** that may contain the data.
* It is an abstract class that defines the programming interface for all input streams that are inherited from it.
* An input stream is automatically opened when we create it. We can explicitly close a stream with the **close( )** method, or let it be closed implicitly when the object is found as a garbage.
* The subclasses inherited from the **InputStream** class can be seen in a hierarchy manner shown below:

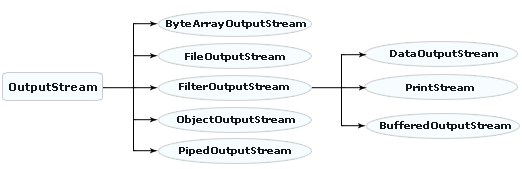


* InputStream is inherited from the Object class. Each class of the InputStreams provided by the java.io package is intended for a different purpose.

|  |  |
| --- | --- |
| Class | Description |
| InputStream | This class represents an input stream of bytes. |
| ByteArrayInputStream | It contains the internal buffer and read data from the stream. |
| FileInputStream | An input stream that reads binary data from a file |
| ObjectInputStream | This class used for recover the object to serialize previously. |
| FilterInputStream | This class overrides all methods of InputStream and contains some other input stream. |
| |  |  | | --- | --- | | DataInputStream | This class reads the primitive data types from the input stream in a machine format. Using readInt(), readByte(), etc methods. | | LineNumberInputStream |  | | BufferedInputStream | This class internally creates buffered array, which is used to read input data. | | PushbackInputStream | It includes the another function of input stream. Such as: "push back" or "unread" one byte. | |  | | | |
| PipedInputStream | In this class the data bytes are written into piped output stream. This class also connected into a piped output stream. |
| SequenceInputStream | It represents the logical concatenation of other input stream. |
| StringBufferInputString |  |

**OutputStream:**

* The OutputStream class is used for writing byte and array of bytes to an output source.
* An output source can be anything such as a file, a string, or memory containing the data.
* Like an input stream, an output stream is automatically opened when we create it. we can explicitly close an output stream with the **close( )** method, or let it be closed implicitly when the object is garbage collected.
* The classes inherited from the **OutputStream** class can be seen in a hierarchy structure shown below:

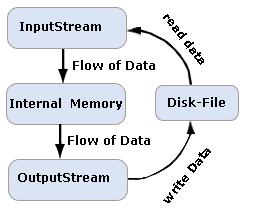


* OutputStream is inherited from the Object class. Each class of the OutputStreams provided by the java.io package is intended for a different purpose.

|  |  |
| --- | --- |
| Class | Description |
| OutputStream | This class represents an output stream of bytes. |
| ByteArrayOutputStream | This class used for data is written into byte array. |
| FileOutputStream | It uses for writing data to a file and also implements an output stream. |
| FilterOutputStream | This class overrides all methods of OutputStream and contains some other output stream. |
| |  |  | | --- | --- | | DataOutputStream | An OutputStream, which is used to write the primitive java datatype to an output stream using writeInt(int n) etc methods | | PrintStream | A output Stream, which is used to print the output of various data types, This class contains print() and println() methods, we use these methods with System.out in java programs | | BufferedOutputStream | This class used for write bytes to output stream. It implements a buffered output stream. | | |
| ObjectOutputStream | This class used to write the primitive data types and also write the object to read by the ObjectInputStream. |
| PipedOutputStream | This class communicates the piped input stream into piped output stream. It creates communication between both. |

**How Streams and Files Work:**

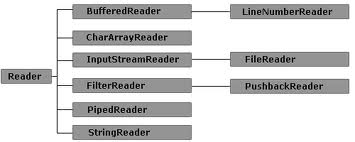
* Java uses **streams** to handle I/O operations through which the data is flowed from one location to another.
* A stream carries data from one location from one place to another.
* For example, an **InputStream** can flow the data from a disk file (may be text file or binary file) to the internal memory and an **OutputStream** can flow the data from the internal memory to a disk file.
* When we work with a text file, we use a **character** stream where one character is treated as per byte on disk.
* When we work with binary files, we use a **byte** stream.



* Byte Streams are used to handle any characters (text), images, audio, video files.
* Character or Text streams can always store and retrieve data in the form of characters (or text) only.

**Reader and Writer:**

* If a class ends with a word Reader or Writer then it is taken as a text stream. Reader reads the text and writer writes the text
* For Ex: FileReader, FileWriter, BufferedReader, BufferedWriter etc.
* If a class name ends with a word Stream, then it is taken as a byte stream. InputStream reads bytes and OutputStream writes bytes.
* For Ex: FileInputStream, FileOuputStream, BufferedInputStream, BufferedOutputStream etc.
* The classes inherited from the **Reader** class can be seen in a hierarchy structure shown below:



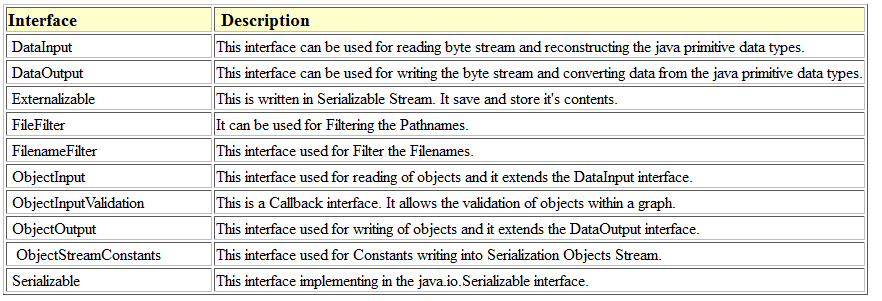
|  |  |
| --- | --- |
| **Class** | **Description** |
| Reader | This class is used for reading character input stream. |
| BufferedReader | This class reads text from character input stream and buffering characters. It also reads characters, arrays and lines. |
| LineNumberReader | An input stream that counts the total number of lines entered. |
| CharacterArrayReader | Used to read data from a character array. |
| InputStreamReader | Translates byte stream to character stream |
| FileReader | This class used for reading characters from a file. |
| FilterReader | It reads the data from the filtered character stream. |
| PushbackReader |  |
| PipedReader | It is a piped character-input stream. |
| StringReader | An input character that reads from a string |

* The classes inherited from the **Writer** class can be seen in a hierarchy structure shown below:

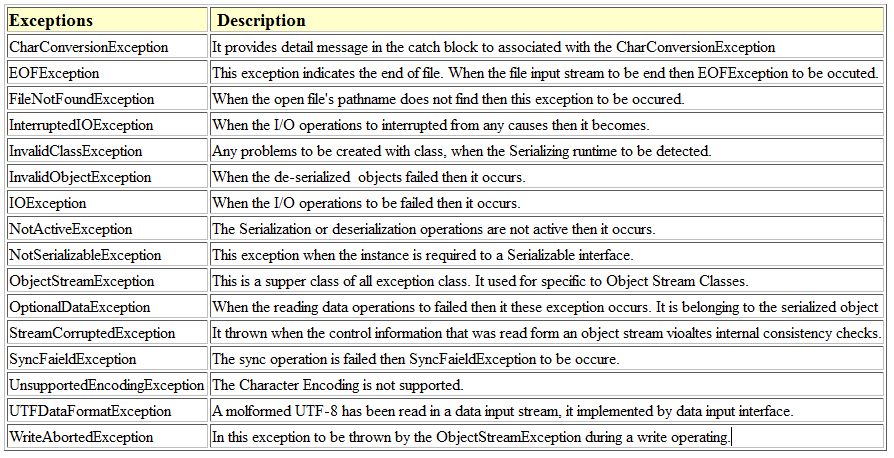
|  |  |
| --- | --- |
| Class | Description |
| Writer | This class is used for writing character to output stream. |
| BufferedWriter | Used to write text to a character output stream |
| CharArrayWriter | Used to write data to a character array |
| FilterWriter | This class used for writing characters to files. |
| OutputStreamWriter | Translates character streams to byte streams |
| FileWriter | Used for writing character files. |
| PipedWriter | It is a piped character-output stream. |
| StringWriter | A character stream that that writes output in a string buffer. |

**Exception Classes and Interfaces of I/O:**

The following summary of **Interfaces** provided by the **java.io** package shown in the table:



The following summary of the exception classes provided by the **java.io** package shown in the table:



**Standard Streams**

* Standard Streams are a feature provided by many operating systems. By default, they read input from the keyboard and write output to the display. They also support I/O operations on files.
* Java also supports three Standard Streams:
* **Standard Input:** Accessed through **System.in** which is used to read input from the keyboard.
* **Standard Output:** Accessed through **System.out**which is used to write output to be display.
* **Standard Error:** Accessed through **System.err** which is used to write error output to be display.

|  |  |
| --- | --- |
| System.out | System.err |
| Both are used to display messages on the monitor | |
| System.out is used to display normal messages as  System.out.println(“hello”); | System.out is used to display error messages as  System.err.println(“this is an error”); |

**Working with files and Directory:**

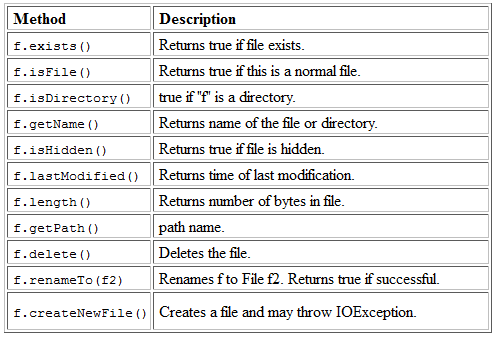
* The **File** class deals with the machine dependent files in a machine-independent manner i.e. it is easier to write platform-independent code that examines and manipulates files using the **File** class.
* This class is available in the **java.lang** package.
* The **java.io.File** is the central class that works with files and directories. The **instance** of this class represents the name of a file or directory on the host file system.
* The **constructors** of the File class are shown in the table:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| File(path) | Create File object for default directory (usually where program is located). |
| File(*dirpath*,*fname*) | Create File object for directory path given as string. |
| File(*dir*, *fname*) | Create File object for directory. |

* Thus the statement can be written as:

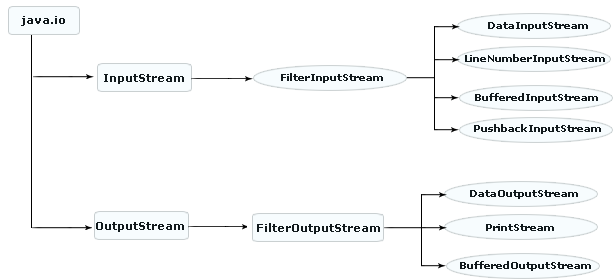
**File f = new File("<filename>");**

* The methods that are used with the file object to get the attribute of a corresponding file shown in the table.



**Filter I/O streams:**

* A filter stream filters data as its being read from or written to the stream.
* **Filter streams** are used to manipulate data reading from an underlying stream.
* **Filter streams** allows the user to make a chain using multiple input stream so that, the operations that are to be applied on this chain, may create a combine effects on several filters.
* By using these streams, there is no need to convert the data from **byte** to **char** while writing to a file. These are the more powerful streams than the other streams of Java.
* The class hierarchy of the Filter streams derived from I/O streams can be shown as:

****

**Serializing An Object:**

* **Serialization** is the process of saving an object in a storage medium (such as a file, or a memory buffer) or to transmit it over a network connection in binary form.
* Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.
* The serialized objects are JVM independent and can be re-serialized by any JVM.
* This process of serializing an object is also called **deflating** or **marshalling** an object.
* Classes **ObjectInputStream** and **ObjectOutputStream** are high-level streams that contain the methods for serializing and deserializing an object.
* The opposite operation of the serialization is called **deserialization** i.e. to extract the data from a series of bytes is s known as **deserialization** which is also called **inflating** or **unmarshalling**.